EPICS device support for Abaco reflective memory

Note that, before setting up this module, you should install the rfm2g, which is the device/driver provided by Abaco.
We confirm that this module works under the condition, CentOS 7.9 (kernel 3.10.0-1160.el7.x86_64) and EPICS 3.15.8.

Download the device support for drvrm5565pci
https://cerldev.kek.jp/trac/EpicsUsersJP/attachment/wiki/epics/pcie5565/drvrm5565pci-221110.tar.gz

---

How to build drvrm5565pci

- Unpack drvrm5565pci-YYMMDD.tar.gz
  $ tar xzf drvrm5565pci-YYMMDD.tar.gz
- Edit configure/RELEASE and modify EPICS_BASE for your condition
  $ cd drvrm5565pci
  $ vi configure/RELEASE
  :
  :
  # EPICS_BASE should appear last so earlier modules can override stuff:
  EPICS_BASE = /opt/epics/R3-15-8/base ← modify here
  :
  :
- Edit drvrm5565App/src/Makefile and include rfm2g in your condition
  #--
  # ADD RULES AFTER THIS LINE
  RELEASE_INCLUDES += -I/usr/lib64/rfm2g ← modify here
- Build it
  $ make

How to build IOC

- Specify the directory of drvrm5565pci on configure/RELEASE

  DRVRM5565 = $(EPICS_BASE)/../modules/drvrm5565pci ← add

- Add drvrm5565pci.dbd at *<prod_name>_DBD* of src/Makefile

  (Following example assumes <prod_name> is rmioc)

  :

  :

  #============================

  # Build the IOC application

  PROD_IOC = rmioc

  •

  •

  # Include dbd files from all support applications:

  rmioc_DBD += drvrm5565pci.dbd ← add

  :

  :

- Add drvrm5565pci at *<prod_name>_LIBS* of src/Makefile

  :

  :

  #============================

  # Build the IOC application

  PROD_IOC = rmioc

  :

  :

  # Add all the support libraries needed by this IOC

  rmioc_LIBS += drvrm5565pci ← add

  :

  :

- Add followings at the last line of src/Makefile

  :

  :

  include $(TOP)/configure/RULES

```
#---------------------------------------
#    ADD RULES AFTER THIS LINE
RELEASE_INCLUDES += -I/usr/lib64/rfm2g     ← add
PROD_DEPLIB_DIRS += /usr/lib64/rfm2g       ← add
EPICS_BASE_IOC_LIBS += rfm2g               ← add
```

- Build it

How to use

Add followings on the start-up script

RM5565PciConfigure(<>card>, <devpath>)

<card> : card number, it should not be overlapped
This number is used also at the EPICS db file.

<devpath> : Specify the device path. Usuary, it is /dev/rfm2g*

Example,

RM5565PciConfigure(0, "/dev/rfm2g0")

How to make db file

How to read the data from the reflective memory with the aai record.

```
record(aai, "pv-name") {
    field(DTYP, "aai5565")
    field(INP,  "#C<card_number> S @<offset>")
    field(FTVL, "<value_type>")
    field(NELM, "<num_of_elems>")
}
```

(The name of device type should be “aai5565”.)

- card_number: The card number designated in argument card in RM5565PciConfigure
- offset: Offset of RM5565 memory
- value_type: Field Type of Value
- num_of_elems: Number of elements

Example

```
record(aai, "RM5565:DEV0:A000:R") {
    field(DTYP, "aai5565")
    field(INP,  "#C0 S @0xA000")
    field(FTVL, "LONG")
    field(NELM, "1")
        .
        .
        .
}
```

Specify the card number and the address offset in the INP field

The card number and address should be specified after “C” and “@”, respectively.

In above case, the 32 bit data are taken out from the address offset 0xA000 of the reflective memory. Then, it is stored in the VAL field.

How to write the data to the reflective memory with the aao record

```
record(aao, "pv-name") {
    field(DTYP, "aao5565")
    field(OUT,  "#C<card_number> S @<offset>,<wait>")
    field(FTVL, "<value_type>")
    field(VAL,  "<data>")
    field(NELM, "<num_of_elems>")
}
```

(The name of device type should be “aao5565”.)

- card_number: The card number designated in argument card in RM5565PciConfigure
- offset: Offset of RM5565 memory
- wait: Waiting time for completion of write (unit in sec)
- value_type: Field Type of Value
- num_of_elems: Number of elements
- data: data to write

Example

```
record(aao, "RM5565:DEV0:A000:W") {
    field(DTYP, "aao5565")
    field(OUT,  "#C0 S @0xA000,1.0")
    field(FTVL, "SHORT")
    field(NELM, "1")
        .
        .
        .
    field(VAL,  "<data>")
}
```

Specify the card number and the address offset in the OUT field

The card number and address should be specified after “C” and “@”, respectively.

In above case, the 32 bit data are written into the address offset 0xA000 of the reflective memory.

Network interruption with the reflective memory

Note, the **inttx** and **intrx** records are used together for the network interruption.

```
record(intrx, "pv-name") {
        field(DTYP, "intrx5565")
        field(SCAN, "I/O Intr")
        field(INP,    "#C<card_number> S<int_type> @")
        field(SNAM, "<subroutine_name>")
}
```

(The name of device type should be “intrx5565”.)

```
record(inttx, "pv-name") {
        field(DTYP, "inttx5565")
        field(OUT,    "#C<card_number> S<int_type> @")
        field(NID,    "<target_node>")
        field(VAL,    "<data>")
}
```

(The name of device type should be “inttx5565”.)

- card_number: The card number designated in argument card in RM5565PciConfigure
- int_type: the Network-interruption type (should be assign 1-4). You can configure the 4 types of the network interruption on the one reflective memory network.
- target_node: the ID for the destination node. (If NID=256, the interruption is delivered all nodes in the network.)
- subroutine_name: the subroutine which are implemented with the node receives the network interruption.
- data: data which are transferred with the interruption

Example

```
record(inttx, "RM5565:DEV0:INT1:TX") {
        field(DTYP, "inttx5565")
        field(OUT,    "#C0 S1 @")
                .
                .
                .
}
record(intrx, "RM5565:DEV1:INT1:RX") {
        field(DTYP, "intrx5565")
        field(SCAN, "I/O Intr")
        field(INP,    "#C0 S1 @")
        field(SNAM, "int1func")
                .
                .
                .
}
```

Above two PVs are configured in the different reflective memory IOC.

When we process “RM5565:DEV0:INT1:TX”,

1. Type-1 network interruption is launched in the network.
2. “RM5565:DEV1:INT1:RX” is processed with the network interruption
3. Then, the subroutine “int1func” is processed.

Error monitoring

The error status can be monitored with the longin record.

record(longin, "pv-name") {
    field(DTYP, "H/W error")
    field(SCAN, "I/O Intr")
    field(INP, "#C<card_number> S @<error_type>[,<severity>]")
}

(The name of device type should be “H/W error”)

- card_number: The card number designated in argument card in RM5565PciConfigure
- error_type:
    BadData ... Bad Data
    FIFOFull ... RX FIFO Full
    RogueDet ... Rogue Packet Detected and Removed
    FIFOAFull ... RX FIFO Almost Full
    SyncLoss ... Sync Loss
- severity
    NO_ALARM ... NO_ALARM
    MINOR ... MINOR_ALARM
    MAJOR ... MAJOR_ALARM
    INVALID ... INVALID_ALARM

Example,

record(longin, "RM5565:DEV0:BadData") {
    field(DTYP, "H/W error")
    field(SCAN, "I/O Intr")
    field(INP, "#C0 S @BadData,MINOR")
        .
        .
        .
}

Specify the card number, error type, and severity in the INP field. In this case, the PV detects the error type “BadData”. The PV is processed and the value in the VAL field is increased when the “BadData” error is occurred.